Available online at www.sciencedirect.com
ScienceDirect
Theoretical Computer Science 370 (2007) 1–18
Theoretical Computer Science
www.elsevier.com/locate/tcs

Longest common subsequence problem for unoriented and cyclic strings

François Nicolas, Eric Rivals*

L.I.R.M.M., U.M.R. 5506, C.N.R.S. – Université de Montpellier II, 161, rue Ada, 34392 Montpellier Cedex 5, France

Received 16 September 2004; received in revised form 3 July 2006; accepted 9 October 2006

Communicated by M. Crochemore

Abstract

Given a finite set of strings X, the LONGEST COMMON SUBSEQUENCE problem (LCS) consists in finding a subsequence common to all strings in X that is of maximal length. LCS is a central problem in stringology and finds broad applications in text compression, conception of error-detecting codes, or biological sequence comparison. However, in numerous contexts, words represent cyclic or unoriented sequences of symbols and LCS must be generalized to consider both orientations and/or all cyclic shifts of the strings involved. This occurs especially in computational biology when genetic material is sequenced from circular DNA or RNA molecules.

In this work, we define three variants of LCS when the input words are unoriented and/or cyclic. We show that these problems are NP-hard, and W[1]-hard if parameterized in the number of input strings. These results still hold even if the three LCS variants are restricted to input languages over a binary alphabet. We also settle the parameterized complexity of our problems for most relevant parameters. Moreover, we study the approximability of these problems: we discuss the existence of approximation bounds depending on the cardinality of the alphabet, on the length of the shortest sequence, and on the number of input sequences. For this we prove that MAXIMUM INDEPENDENT SET in r-uniform hypergraphs is W[1]-hard if parameterized in the cardinality of the sought independent set and at least as hard to approximate as MAXIMUM INDEPENDENT SET in graphs.

Keywords: Longest common subsequence; LCS; Cyclic string; Sequence comparison; Pattern recognition; Graph; Hypergraph; Maximum stable set; Maximum independent set; Approximation; Parameterized complexity; NP-hard; W[1]-hard

1. Introduction

1.1. Longest common subsequence and shortest common supersequence

Let s, t be two words and denote the length of a word s by $|s|$. s is a *subsequence* of t if s can be obtained by erasing zero or more symbols of t. In that case, t is a *supersequence* of s. For a set of words X, finding a word s

* Corresponding address: LIRMM – CNRS, Computer Science Department, 161, rue Ada, F-34392 Montpellier Cedex 5, France. Tel.: +33 4 67 41 86 64; fax: +33 4 67 41 85 00.
E-mail address: rivals@lirmm.fr (E. Rivals).
URL: http://www.lirmm.fr/~rivals (E. Rivals).

0304-3975/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2006.10.002

that is a subsequence common to all words in X and such that $|s|$ is maximal is known as the LONGEST COMMON SUBSEQUENCE problem (LCS). The dual problem of finding a SHORTEST COMMON SUPERSEQUENCE for the input X is denoted SCS. The LCS problem, as well as the SCS problem, find numerous applications, for instance in string correction, text compression, conception of error-detecting codes, biological sequence comparison, and many more (see [1] for a review). Moreover, it is of theoretical interest as a central problem in stringology and as a simpler version of the more general MULTIPLE ALIGNMENT problem [2, Chapter 14].

1.2. Examples of cyclic and/or unoriented words

We review some domains in which linear words represent cyclic and/or unoriented sequences of symbols.

1.2.1. In computational biology

In nature, inherited information is stored on linear or circular DNA or RNA molecules [3]. Bacterial genomes are in the majority circular, as are chloroplastic and mitochondrial genomes. Viruses often store their genetic material on circular DNA, especially bacteriophages, which are widely used as vectors for cloning a gene of another species. Plasmids, small circular DNA molecules that have the ability to replicate on their own, are extensively used in biotechnology [3]. All such cyclic molecules are sequenced and represented as linear strings by choosing an arbitrary starting point. It follows that the comparison of two such sequences needs to consider all possible *cyclic shifts* (also called *conjugates*) of one of the sequences. This is cyclic string comparison.

DNA, as well as RNA, are oriented molecules. The double stranded nature of DNA relies on the complementarity of the nucleotides, which enables the hybridization of the two strands: Adenine binds to Thymine and vice versa ($A \leftrightarrow T$), while Guanine binds to Cytosine and vice versa ($G \leftrightarrow C$). As the DNA strands have inverse orientation, it follows that their sequences are reverse and complementary. In the sequencing process, which is now carried out automatically, the DNA of interest is cloned into a double-stranded vector and the vector is sequenced. For technical reasons, it is not always possible to know in which orientation the DNA is inserted into the vector (*i.e.*, on which strand). Therefore, in some cases, *e.g.*, for Expressed Sequence Tags, sequences must be compared in each orientation [4]. We call this a comparison up to a complementary reversal. In this article, we consider only the matter of orientation and leave aside the complementarity, since the complexity of the various LCS problems remains essentially the same.

1.2.2. In pattern recognition

Another domain in which cyclic strings arise is pattern representation and recognition [5–7]. There, the closed contour of a two-dimensional (polygonal) shape is encoded into a linear string by choosing arbitrarily a start position on the contour. Determining if two shapes are similar or equal requires one to compare one string with all cyclic shifts of the other. Practically, this type of comparison is applied, for instance in an industrial context, to recognize the class of an object currently visible on a conveyor belt. In this context, if the side on which the object is laid is chosen randomly, the contour may be encoded in either direction; it is thus necessary to perform an unoriented and cyclic comparison.

1.3. Known results

A large literature is devoted to LCS and to SCS; we summarize below the main results.

1.3.1. Comparison of two words

Let s and t be two words with $|t| \leq |s|$.

1.3.1.1. Linear case. A well-known algorithm computes a longest common subsequence (lcs) of s and t in $O(|s||t|)$ time and $O(|t|)$ space by dynamic programming [8]. The time bound has been improved to $O(|s||t| / \log |s|)$ time [9]. The original algorithm has been generalized to compute an optimum alignment between s and t with unrestricted cost matrices and unbounded alphabet within the same time and space bounds [2, Chapter 12]. For unit costs, one can compute an optimum alignment between s and t in $O(e|s|)$ time where e is the optimum alignment cost (*i.e.*, the Levenshtein distance between s and t) [10]. When the alphabet is bounded, the complexity of $O(|s||t| / \log |s|)$ can also be achieved to compute an optimum alignment between s and t with unrestricted cost matrices [11].

1.3.1.2. Cyclic case. Define a *cyclic alignment* between s and t, as an alignment between a cyclic shift of s and a cyclic shift of t. The problem of finding an optimum cyclic alignment between s and t has been studied. Little thinking shows that it reduces to finding an optimum alignment between s and any cyclic shift of t. The application of dynamic programming to compute optimum alignments of all combinations yields a brute-force algorithm in $O\left(|s||t|^2\right)$ time. Maes [12] exhibits an $O(|s||t|\log|t|)$ time algorithm. Schmidt [13] gives an $O(|s||t|)$ time algorithm if there exists two positive integers S and M such that all insertions, deletions and substitutions cost S, and all matches cost $-M$. For unit costs, one can compute an optimum cyclic alignment between s and t in $O(e|s|)$ time, where e is the optimum cyclic alignment cost [14].

1.3.2. LCS *in the case of many input strings*

LCS can be solved in polynomial time when the number of input strings is fixed [15]. However, when the number of input strings is unbounded, LCS becomes intractable.

- Even for binary alphabets, the decision problem associated with LCS is NP-hard [16], and W[1]-hard when parameterized in the number of input strings [17].
- For unbounded alphabets, LCS is hard to approximate. More precisely, approximating LCS is at least as hard as approximating MAXIMUM INDEPENDENT SET [18].

The parameterized complexity with respect to several other parameters is also studied in [19,20], while the average error of approximation algorithms is also investigated in [18]. More details are given in Section 2.3.5.

Complexity and approximability of the variants of LCS where input strings are cyclic and/or unoriented remain open.

1.3.3. Complexity of SCS

1.3.3.1. Linear words. For two input strings, a shortest common supersequence can be deduced from a lcs in linear time. For a bounded number of input strings, SCS remains polynomial. However, for many input strings, the problem is NP-complete [16]. It remains true even if the input alphabet is binary [21] or if all input words have length two [22].

When parameterized by the number of input strings, SCS is W[1]-hard even for binary alphabets [17].

If the input alphabet is bounded then SCS is approximable within a constant ratio. Otherwise, it does not admit an approximation algorithm of constant ratio unless P = NP [18].

1.3.3.2. Cyclic words. The variant of SCS for cyclic strings, denoted SCCS, was shown to be NP-complete even if restricted to binary alphabet, or to input words of length three [23]. Conversely to SCS, SCCS is polynomial when all input words have length two [23].

1.4. Our contribution

In this work, we define three variants of LCS when considering a set of input words that are (i) unoriented, (ii) cyclic, or (iii) both.

(1) We show that these problems are NP-hard, and W[1]-hard with respect to the number of input strings, even when restricted to instances over a binary alphabet (Section 3).
(2) We study their approximability when the input alphabet is unbounded and conclude that they are at least as hard to approximate as MAXIMUM INDEPENDENT SET (Section 4.2). We also prove some new results on the complexity of the latter (Section 4.1).
(3) As a by-product of our reductions, we settle the parameterized complexity of our three problems for most relevant parameters. Our results are summarized in Section 5.

Section 2 below gives basic notations, defines the problems and summarizes useful known results on MAXIMUM INDEPENDENT SET and on LCS. We conclude and list open questions in the final section.

2. Notations and definitions

We denote by $\mathbb{N}$ the set of non-negative integers. For any $a, b \in \mathbb{N}$, $[a, b]$ denotes the set $\{n \in \mathbb{N} : a \leq n \leq b\}$. For any finite set X, we denote by $\#X$ the *cardinality* of X. Throughout this paper, ϵ denotes an arbitrarily small positive real constant.

2.1. Approximability of maximization problems

Let MAX be a *maximization* problem. To any *instance* x of MAX is associated a set $S(x)$ of feasible *solutions*. Let (x, s) be a pair where x is an instance of MAX and $s \in S(x)$ a solution. To each such pair (x, s) is associated a *measure* $\mu(x, s) \in \mathbb{N}$ of the quality of the solution s for the instance x. An *optimal solution* of MAX on x is a solution $s^* \in S(x)$ with measure $\mu(x, s^*) = \max_{s \in S(x)} \mu(x, s)$.

Let ρ be a function that maps an instance of MAX to a real number greater than or equal to 1. An *approximation algorithm with bound* ρ for MAX is a polynomial algorithm that for each input instance x of MAX returns a solution $a \in S(x)$ satisfying $\rho(x)\mu(x, a) \geq \max_{s \in S(x)} \mu(x, s)$. We say that MAX is *approximable within bound* ρ if there exists such an algorithm for the problem MAX.

The *decision problem associated with* MAX, denoted by MAX_D, is stated as follows:

Name: MAX_D

Instance: A pair (x, k) where
- x is an instance of MAX, and where
- k is a non-negative integer called the *acceptance threshold*.

Question: Does there exist $s \in S(x)$ satisfying $\mu(x, s) \geq k$?

Throughout this paper, acceptance thresholds are always denoted k.

2.2. Independent sets in graphs and hypergraphs

2.2.1. Definitions

A *hypergraph* is a pair $H = (\text{V}(H), \mathcal{E}(H))$ where $\text{V}(H)$ is a finite set and $\mathcal{E}(H)$ is a set of subsets of $\text{V}(H)$. The elements of $\text{V}(H)$ are the *vertices* of H and the ones of $\mathcal{E}(H)$ are the *hyperedges* of H. We denote by $|H| := \#\text{V}(H)$ the number of vertices in H. A hypergraph *on* V, where V is any finite set, is a hypergraph H such that $\text{V}(H) = V$.

An *independent set* of H is a set of vertices of H that fully contains no hyperedge of H. In the literature, an independent set is also termed a *stable*. We denote by $\alpha(H)$ the *independence number* or *stability* of H, that is the maximum cardinality of an independent set of H, and define the problem:

Name: MAXIMUM INDEPENDENT SET IN HYPERGRAPHS (MISH)

Instance: A hypergraph H on $[1, |H|]$.

Solution: An independent set I of H.

Measure: The cardinality of I.

A *r-uniform* hypergraph (where $r \in \mathbb{N} \setminus \{0, 1\}$) is a hypergraph whose hyperedges have cardinality r. A *graph* is a 2-uniform hypergraph. The hyperedges of a graph are called *edges*. For any $r \in \mathbb{N} \setminus \{0, 1\}$, we denote by r-MISH the restriction of MISH to r-uniform hypergraphs, H. The 2-MISH problem is usually called MAXIMUM INDEPENDENT SET or MAXIMUM STABLE SET, and its instances, which are graphs, are denoted by G instead of H.

2.2.2. Known results

The decision problem 2-MISH_D associated with 2-MISH is

- NP-complete [24], and
- W[1]-complete when parameterized by its acceptance threshold, k (*i.e.*, the cardinality of the sought independent set) [25].

The maximization problem MISH is approximable within bound $O(|H|/\log|H|)$ [26]. In the case of 2-MISH, there exists an approximation algorithm with bound $O\big(|G|/\log^2|G|\big)$ [27]. Note that the trivial algorithm that for each

input hypergraph H returns an independent set with cardinality 1 yields bound $|H|$ for MISH, which is only slightly worse than the previous ones.

Let δ be a mapping that maps $\mathbb{N}$ to the subset of reals between 0 and 1, inclusive. Further studies on the approximability of 2-MISH show that the existence of an approximation algorithm with bound $|G|^{\delta(|G|)}$ implies the improbable inclusion of NP in various complexity classes according to the asymptotic behavior of δ:

- 2-MISH is not approximable within bound $|G|^{0.5-\epsilon}$, unless NP = P [28].
- 2-MISH is not approximable within bound $|G|^{1-\epsilon}$, unless NP = ZPP [28].
- 2-MISH is not approximable within bound $|G|^{1-O((\log\log|G|)^{-0.5})}$, unless NP $\subseteq$ ZPTIME$\left(2^{O((\log n)(\log\log n)^{1.5})}\right)$ [29].

2.3. Words, subsequences and languages

2.3.1. Words and languages

An *alphabet* Σ is a finite set of *letters*. A *word* over Σ is a finite sequence of elements of Σ. The set of all words over Σ is denoted by $\Sigma^\star$. For a word x, $|x|$ denotes the length of x. Given two words x and y, we denote by xy the *concatenation* of x and y. For every $n \in \mathbb{N}$, we denote by x^n the nth *power* of x, that is, the concatenation of n copies of x (note that x^0 is the empty word). A word x is *unary* if there exists a letter a such that $x = a^{|x|}$.

For every $i \in [1, |x|]$, $x[i]$ denotes the ith letter of x: $x = x[1]x[2]\cdots x[|x|]$. For every letter a, $|x|_a := \#\{i \in [1, |x|] : x[i] = a\}$ denotes the number of *occurrences* of the letter a in x.

The *mirror image* (also called *reversal*) of x is the word $\tilde{x} := x[|x|]\cdots x[2]x[1]$. Two words x and y are *conjugates* of each other if there exist two words u and v such that $x = uv$ and $y = vu$.

Example 1. The mirror images of `abcd` and `ababa` are `dcba` and `ababa` respectively. The conjugates of `abcd` are `abcd`, `bcda`, `cdab`, and `dabc`. The conjugates of `ababab` are `ababab` and `bababa`.

A *language* (over Σ) is any set X of words (over Σ). We denote by $\sigma(X)$ the cardinality of the smallest alphabet Σ such that $X \subseteq \Sigma^\star$. We say that X is *unary* if $\sigma(X) = 1$, and that X is *binary* if $\sigma(X) \leq 2$.

2.3.2. Subsequences of a word

In addition to the usual notion of subsequence, we define three more types of subsequences.

Definition 2 (*Subsequence, U-, C-, UC-subsequence*). Let s and x be two words. We say that:

(1) s is a *subsequence* of x if one obtains s by erasing some letters of x (eventually all or none),
(2) s is an *unoriented subsequence* (U-subsequence) of x when s or $\tilde{s}$ is a subsequence of x,
(3) s is a *cyclic subsequence* (C-subsequence) of x when a conjugate of s is a subsequence of x,
(4) s is an *unoriented cyclic subsequence* (UC-subsequence) of x when a conjugate of s is a U-subsequence of x.

Remark 3. For any words x and s one has:

- s is a U-subsequence of x iff s is a subsequence of x or of $\tilde{x}$,
- s is a C-subsequence of x iff s is a subsequence of a conjugate of x,
- s is a UC-subsequence of x iff s is a U-subsequence of a conjugate of x,
- any subsequence of x is both a U-subsequence and a C-subsequence of x,
- any U-subsequence of x and any C-subsequence of x are UC-subsequences of x.

2.3.3. Common subsequences of a language

Let X be a non-empty language. A *common* subsequence (resp. U-subsequence, resp. C-subsequence, resp. UC-subsequence) of X is a word that is a subsequence (resp. U-subsequence, resp. C-subsequence, resp. UC-subsequence) of each word in X.

Definition 4 (lcs, lcus, lccs, lcucs). For any non-empty language X, we denote by $\mathrm{lcs}(X)$ (resp. $\mathrm{lcus}(X)$, resp. $\mathrm{lccs}(X)$, resp. $\mathrm{lcucs}(X)$) the length of a longest common subsequence (resp. U-subsequence, resp. C-subsequence, resp. UC-subsequence) of X.

Table 1
Parameterized complexity of the LCS problem

$\#X$	$\sigma(X)$	k	LCS_D
–	≤ 2	–	NP-complete [16]
Bounded	–	–	Polynomial [15]
Parameter	≤ 2	–	W[1]-hard [17]
–	Parameter	Parameter	F.P.T. (see Section 2.3.5.1)
–	–	Parameter	W[2]-hard [20]
Parameter	–	Parameter	W[1]-complete [20]
Parameter	Parameter	–	W[t]-hard $\forall t \geq 1$ [19]

Remark 5. For any non-empty language X, it follows from Remark 3 that

$$\mathrm{lcs}(X) \leq \mathrm{lcus}(X) \leq \mathrm{lcucs}(X) \quad \text{and} \quad \mathrm{lcs}(X) \leq \mathrm{lccs}(X) \leq \mathrm{lcucs}(X).$$

Example 6. One easily checks in the following examples that each of these inequalities happens to be strict, which guarantees the relevance of Definition 4:
• $\mathrm{lcs}(X) = 1 < 2 = \mathrm{lcus}(X) = \mathrm{lccs}(X) = \mathrm{lcucs}(X)$ for $X := \{\mathtt{01}, \mathtt{10}\}$,
• $\mathrm{lcus}(Y) = 2 < 3 = \mathrm{lccs}(Y) = \mathrm{lcucs}(Y)$ for $Y := \{\mathtt{011}, \mathtt{101}\}$,
• $\mathrm{lccs}(Z) = 5 < 6 = \mathrm{lcus}(Z) = \mathrm{lcucs}(Z)$ for $Z := \{\mathtt{101001}, \mathtt{100101}\}$.
The examples also illustrate that lccs and lcus are not comparable.

2.3.4. Definitions of the problems

To each of these notions of common subsequence corresponds an optimization problem. We call respectively

- Longest Common Subsequence (LCS),
- Longest Common Unoriented Subsequence (LCUS),
- Longest Common Cyclic Subsequence (LCCS), and
- Longest Common Unoriented Cyclic Subsequence (LCUCS)

the following maximization problems:

Name: LCS (resp. LCUS, resp. LCCS, resp. LCUCS)
Instance: A non-empty finite language X.
Solution: A common subsequence (resp. U-subsequence, resp. C-subsequence, resp. UC-subsequence) s of X.
Measure: The length of s.

2.3.5. Known results about the LCS *problem*

2.3.5.1. Classical and parameterized complexity. The parameterized complexity of LCS_D has been studied for any combination of the three following parameters:

- the number of input words, $\#X$,
- the cardinality of the input alphabet, $\sigma(X)$, and
- the acceptance threshold, k (*i.e.*, the length of the sought subsequence).

Known results are summarized in Table 1. In accordance with this table, LCS_D, LCCS_D, LCUS_D, and LCUCS_D are obviously F.P.T. for the aggregate parameter $(\sigma(X), k)$. Indeed, one can enumerate all $(\sigma(X))^k$ words of length k over the input alphabet and check them against each word in X.

2.3.5.2. Approximability. Let b be a mapping that maps $\mathbb{N} \times \mathbb{N} \times \mathbb{N}$ to the subset of reals greater than or equal to 1. If LCS is approximable within bound $b(\min_{x \in X} |x|, \sigma(X), \#X)$ then 2-MISH is approximable within bound $b\big(|G|, |G|, 2|G|\big)$ [18].

3. Intractability of LCUS, LCCS, and LCUCS

In this section, we consider the decision problems LCUS_D, LCCS_D, and LCUCS_D associated with LCUS, LCCS, and LCUCS respectively. We demonstrate that these problems are

- NP-complete (Theorem 14(i)), and
- W[1]-hard with respect to the number of input words, $\#X$ (Theorem 14(ii)).

Moreover, both results hold even if the problems are restricted to binary input languages (*i.e.*, languages X such that $\sigma(X) \leq 2$). We also settle the parameterized complexity of our LCS_D variants for parameter $(\#X, \sigma(X))$ (Theorem 14(iii)).

However, note that, as LCS, each of the three problems LCUS, LCCS, and LCUCS is tractable when the number of input words is bounded.

Proposition 7. LCCS_D, LCUS_D, *and* LCUCS_D *are polynomial when* $\#X$ *is fixed.*

Proof. By dynamic programming, LCS can be solved in $O\big((\#X)\prod_{x\in X}|x|\big)$ time [15]. For any language X, there are $2^{\#X}$ combinations of the words in X with a fixed orientation. Therefore, using the above mentioned algorithm for LCS on each combination, and returning the longest common subsequence found, solves LCUS in $O\big((\#X)2^{\#X}\prod_{x\in X}|x|\big)$ time, which is polynomial when $\#X$ is fixed.

A similar exploration algorithm considering all possible cyclic shifts of the words in X can be used to solve LCCS in $O\big((\#X)\prod_{x\in X}|x|^2\big)$ time. Again, considering all possible cyclic shifts and all possible orientations of the words in X yields an algorithm for LCUCS that requires $O\big((\#X)2^{\#X}\prod_{x\in X}|x|^2\big)$ time. These running times are also polynomial when $\#X$ is fixed. □

We now turn to the proof of the intractability results announced at the beginning of the section. All proofs rely on a reduction from LCS. This reduction is described in Definition 9. As explained in Lemmas 8 and 10, we use a "padding argument" to ensure synchronization.

Lemma 8 (*Synchronization Lemma*). *Let a, b be two distinct letters, and let s, t be two words. Let m, n, p, q be four integers with $m, n > 0$, and $p, q > |s|$. Set $u := a^m s b^n$ and $v := a^p t b^q$. Then, v or $\tilde{v}$ is a conjugate of u iff $u = v$.*

Proof. Any conjugate of u that is *distinct from* u adopts one of the three following forms:

(i) $a^j s b^n a^{m-j}$ with $j \in [1, m-1]$,
(ii) $b^j a^m s b^{n-j}$ with $j \in [1, n-1]$, or
(iii) $s_2 b^n a^m s_1$ where s_1 and s_2 are two words such that $s = s_1 s_2$.

• Assume that v is a conjugate of u. The word v ends by letter b, so it is not of the form (i), and v begins with letter a, so it is not of the form (ii). Moreover, as $|s|$ is less than p, v admits $a^{|s|+1}$ as a prefix, which is not the case of words of the form (iii). Thus, v is a conjugate of u which is not of any of these three forms. The only possibility left is $u = v$.

• Let us suppose that $\tilde{v}$ is a conjugate of u. As $\tilde{v} = b^q \tilde{t} a^p$ begins by letter b, $\tilde{v}$ is distinct from u and does not adopt the form (i). Moreover as $\tilde{v}$ ends by letter a, it does not adopt the form (ii) either. It follows that $\tilde{v}$ has the form (iii) and thus, there exists two words s_1, s_2 such that $s = s_1 s_2$ and $b^q \tilde{t} a^p = s_2 b^n a^m s_1$. Since by hypothesis $q \geq |s| \geq |s_2|$, one has $s_2 = b^{|s_2|}$ and similarly, $p \geq |s| \geq |s_1|$ compels $s_1 = a^{|s_1|}$. We obtain that $\tilde{v} = b^{|s_2|+n} a^{m+|s_1|}$ and $u = a^m s_1 s_2 b^n = a^{m+|s_1|} b^{|s_2|+n}$, which yields $u = v$. □

Definition 9. For any finite language X and any distinct letters a, b, we define:

$$M_X := \max_{x\in X}|x| \quad \text{and} \quad X^{\mathrm{UC}}_{a,b} := \big\{a^{2M_X+1} x b^{2M_X+1} : x \in X\big\}.$$

The synchronization lemma enables us to prove the fundamental property of our gadget:

Lemma 10.

$$\mathrm{lcucs}(X^{\mathrm{UC}}_{a,b}) = \mathrm{lccs}(X^{\mathrm{UC}}_{a,b}) = \mathrm{lcus}(X^{\mathrm{UC}}_{a,b}) = \mathrm{lcs}(X^{\mathrm{UC}}_{a,b}) = \mathrm{lcs}(X) + 4M_X + 2.$$

Proof. One easily checks the following property.

Claim 11. *For any non-empty language W and any letter c, one has*

$$\mathrm{lcs}(Wc) = \mathrm{lcs}(cW) = \mathrm{lcs}(W) + 1$$

where $Wc = \{wc : w \in W\}$ *and* $cW = \{cw : w \in W\}$.

Repeatedly applying Claim 11, we get

$$\mathrm{lcs}(X^{\mathrm{UC}}_{a,b}) = \mathrm{lcs}(X) + 4M_X + 2.$$

It remains to show that $\mathrm{lcucs}(X^{\mathrm{UC}}_{a,b}) = \mathrm{lccs}(X^{\mathrm{UC}}_{a,b}) = \mathrm{lcus}(X^{\mathrm{UC}}_{a,b}) = \mathrm{lcs}(X^{\mathrm{UC}}_{a,b})$ or equivalently, according to Remark 5, that $\mathrm{lcucs}(X^{\mathrm{UC}}_{a,b}) \leq \mathrm{lcs}(X^{\mathrm{UC}}_{a,b})$.

Claim 12. *We say that two words u and v are* conjugate up to a reversal *if v or $\tilde{v}$ is a conjugate of u. The relation of conjugacy up to a reversal is an equivalence relation.*

This claim is easily deduced from the fact that (usual) conjugacy is an equivalence relation [30].

Let v be a common UC-subsequence of $X^{\mathrm{UC}}_{a,b}$ of maximal length $\mathrm{lcucs}(X^{\mathrm{UC}}_{a,b})$. For any $x \in X$, there exists a subsequence u_x of $a^{2M_X+1}xb^{2M_X+1}$ such that v or $\tilde{v}$ is a conjugate of u_x. Consider the family of words $(u_x)_{x\in X}$; for all $x, y \in X$, we have

- $|u_x| = \mathrm{lcucs}(X^{\mathrm{UC}}_{a,b})$ because u_x has the same length as v,
- u_x is a subsequence of $a^{2M_X+1}xb^{2M_X+1}$ by definition,
- u_y or $\tilde{u}_y$ is a conjugate of u_x, by Claim 12.

Claim 13. $\forall x, y \in X, u_x = u_y$.

Proof. Words u_x and u_y can be written as $u_x = a^m s b^n$ and $u_y = a^p t b^q$ with s (resp. t) being a subsequence of x (resp. y), and $m, n, p, q \in [0, 2M_X + 1]$. By contradiction, assume $p \leq M_X$; then we would have

$$\begin{aligned}\mathrm{lcucs}(X^{\mathrm{UC}}_{a,b}) = |u_y| = p + |t| + q &\leq M_X + |y| + 2M_X + 1\\ &\leq 4M_X + 1\\ &< 4M_X + 2 + \mathrm{lcs}(X) = \mathrm{lcs}(X^{\mathrm{UC}}_{a,b})\end{aligned}$$

which, by Remark 5, is a contradiction.

We know now that $p > M_X \geq |x| \geq |s|$ and $q > |s|$ for the same reasons. Symmetrically, m and n are larger than M_X, and thus they are positive. Therefore, Lemma 8 with $(u, v) := (u_x, u_y)$ applies and we get $u_x = u_y$. This concludes the proof of Claim 13. □

We can now conclude the proof of Lemma 10. By Claim 13, the u_x's ($x \in X$) are all equal to the same word u, which is a common subsequence of $X^{\mathrm{UC}}_{a,b}$. (u can be written as $a^{2M_X+1}wb^{2M_X+1}$ with w a longest common subsequence of X.) Therefore, $\mathrm{lcs}(X^{\mathrm{UC}}_{a,b}) \geq |u| = \mathrm{lcucs}(X^{\mathrm{UC}}_{a,b})$, which we wanted. □

We can now easily prove the main result of this section.

Theorem 14. *The problems* LCUCS_D, LCCS_D, *and* LCUS_D *are*

(i) NP-*complete even if restricted to binary input languages,*
(ii) W[1]-*hard for parameter* $\#X$ *even if restricted to binary input languages,*
(iii) W[t]-*hard for parameter* $(\#X, \sigma(X))$ *for any* $t \in \mathbb{N} \setminus \{0\}$.

Proof. First, note that the decision problems LCUCS_D, LCCS_D, and LCUS_D are in NP because LCUCS, LCCS, and LCUS are NP-optimization problems (see [31] for a definition of this complexity class). Indeed, given a word s, one can check in polynomial time whether s is a common U-subsequence (resp. C-subsequence, resp. UC-subsequence) of X.

Let us now prove the intractability results. Let X be a non-empty, finite language and let $k \in \mathbb{N}$. If the input language X is unary, then the problems are polynomial since $\mathrm{lcs}(X) = \mathrm{lcus}(X) = \mathrm{lccs}(X) = \mathrm{lcucs}(X) = \min_{x\in X} |x|$.

Now, let us assume that $\sigma(X) \geq 2$ and compute a pair of distinct letters (a_X, b_X) such that each letter occurs in at least a word of X. By Lemma 10, the function

$$(X, k) \longmapsto (X^{\mathrm{UC}}_{a_X,b_X}, k + 4M_X + 2)$$

is a valid M-reduction from LCS_D to LCUS_D, LCCS_D, and LCUCS_D (an M-reduction is a many-to-one reduction). Moreover, our reduction is computable in polynomial time and preserves:

- the number of input words since $\#X^{\mathrm{UC}}_{a_X,b_X} = \#X$, and
- the input alphabet size since $\sigma(X^{\mathrm{UC}}_{a_X,b_X}) = \sigma(X)$.

This enables us to generalize to LCUS_D, LCCS_D, and LCUCS_D some intractability results established so far for LCS_D: NP-completeness even when $\sigma(X) = 2$ in [16], W[1]-hardness for parameter $\#X$ even when $\sigma(X) = 2$ in [17], and W[t]-hardness for parameter $(\#X, \sigma(X))$ for any $t \in \mathbb{N} \setminus \{0\}$ in [19]. □

4. Approximability of MISH, LCS, LCUS, LCCS, and LCUCS

In this section, we first study the approximability of the MAXIMUM INDEPENDENT SET problem in hypergraphs. These results in hand, we can prove some theorems on the hardness to approximate our LCS variants. This section is divided in two. In Section 4.1, we consider the MAXIMUM INDEPENDENT SET problem in r-uniform hypergraphs (r-MISH) for $r \in \mathbb{N} \setminus \{0, 1, 2\}$. We show that

- r-MISH_D is W[1]-hard with respect to its acceptance threshold (Theorem 18(i)), and that
- r-MISH is at least as hard to approximate as 2-MISH (Theorem 18(ii)).

In Section 4.2, we study the approximability of LCS, LCUS, LCCS, and LCUCS when the input alphabet is unbounded. We generalize to LCUS, LCCS, and LCUCS the (loose) approximation bounds previously established for LCS (Section 4.2.1). We also show that

- LCS and LCUS are at least as hard to approximate as 2-MISH (Section 4.2.2.1), and that
- LCCS and LCUCS are at least as hard to approximate as 3-MISH (Section 4.2.2.2).

As a by-product, we settle the parameterized complexity of the decision problems associated to our three LCS variants with respect to their acceptance thresholds.

4.1. *Inapproximability of* MISH

Throughout this section, r denotes an element of $\mathbb{N} \setminus \{0, 1\}$. Before proving the main result (Theorem 18) we need several lemmas.

Lemma 15. *Let $k_0 \in \mathbb{N}$. The decision problem* 2-MISH_D *remains* W[1]*-hard when parameterized in its acceptance threshold, denoted k, even if restricted to instances (G, k) whose acceptance threshold k is at least k_0.*

Proof. Let (G, k) be an instance of 2-MISH_D. We build another instance $(\hat{G}, \hat{k})$ of 2-MISH_D as follows: $\hat{G}$ is obtained by adding k_0 isolated vertices to G, and we set $\hat{k} := k + k_0$. Note that $\hat{k} \geq k_0$. Clearly, G admits an independent set of cardinality k if and only if $\hat{G}$ admits an independent set of cardinality $\hat{k}$. As our reduction preserves the parameter, our result follows from the W[1]-hardness of the general 2-MISH_D [25]. □

Lemma 16. *For any $p \in \mathbb{N}$, there exists a mapping computable in polynomial time that maps any hypergraph H to an independent set of H, denote it J^p_H, having cardinality* $\min\{\alpha(H), p\}$.

Proof. Enumerate all $O(|H|^p)$ subsets of $\mathrm{V}(H)$ whose cardinality is at most p and memorize those that are independent sets of H. Among these selected subsets return one of maximal cardinality. For fixed p, this takes polynomial time. □

Lemma 17. *Let $k_0 \in \mathbb{N}$ and let ρ be a mapping that, to a r-uniform hypergraph H, associates a real $\rho(H) \geq 1$.*

We consider the restriction of r-MISH *to r-uniform hypergraphs H satisfying $\alpha(H) / \rho(H) \geq k_0$. Let us assume that this restriction is approximable within bound $\rho(H)$. Then, the general* r-MISH *problem is approximable within bound $\rho(H)$.*

Proof. Let us denote by $\mathcal{H}^r_{k_0,\rho}$ the class of r-uniform hypergraphs H satisfying $\alpha(H) \,/\, \rho(H) \geq k_0$. Let A be an approximation algorithm with bound $\rho(H)$ for the restriction of r-MISH to $\mathcal{H}^r_{k_0,\rho}$. Let f be a polynomial such that, for any input $H \in \mathcal{H}^r_{k_0,\rho}$, Algorithm A stops after at most $f(|H|)$ steps.

Consider the following algorithm B. For any input r-uniform hypergraph H, B first runs Algorithm A on H and counts its number of steps.

(i) If before $f(|H|)$ steps algorithm A returns an independent set I of H of cardinality at least k_0, then B returns I.
(ii) Otherwise (*i.e.*, when after $f(|H|)$ steps A did not stop or returned something else than an independent set of H of cardinality at least k_0), B returns $J^{k_0}_H$ (as defined in Lemma 16).

Algorithm B takes polynomial time since by Lemma 16 case (ii) requires polynomial time. It remains to show that the approximation bound of B is $\rho(H)$.

• First, assume that $H \in \mathcal{H}^r_{k_0,\rho}$. Algorithm A applied to H stops after at most $f(|H|)$ steps and, by hypothesis, returns an independent set I of H of cardinality $\#I \geq \alpha(H) \,/\, \rho(H) \geq k_0$. Then, Algorithm B falls in case (i) and returns I, which satisfies the desired bound.

• Now assume that $H \notin \mathcal{H}^r_{k_0,\rho}$. In case (i), B returns an independent set of H of cardinality at least k_0, while in case (ii), B returns an independent set of H of cardinality $\min\{\alpha(H), k_0\}$. In both cases, B returns an independent set of H of cardinality at least $\min\{\alpha(H), k_0\}$. One has $k_0 > \alpha(H) \,/\, \rho(H)$ by hypothesis, and $\alpha(H) \geq \alpha(H) \,/\, \rho(H)$ since $\rho(H) \geq 1$. Thus, $\min\{\alpha(H), k_0\} \geq \alpha(H) \,/\, \rho(H)$, which we wanted. □

Theorem 18. *Let $r \in \mathbb{N} \setminus \{0, 1\}$, $k_0 \in \mathbb{N}$, and let b be a mapping that to any integer associates a real greater than or equal to* 1.

(i) *The decision problem r-MISH$_D$ is* W[1]*-hard when parameterized in its acceptance threshold, k. Moreover, this result still holds even if r-MISH$_D$ is restricted to instances (H, k) whose acceptance threshold k is at least k_0.*
(ii) *If r-*MISH *is approximable within bound $b(|H|)$ then* 2-MISH *is approximable within bound $b(|G|)$.*

Proof. For any graph G, consider the set of r-subsets of $\mathrm{V}(G)$ that contains at least an edge of G. We denote by H^r_G the r-uniform hypergraph on $\mathrm{V}(G)$ whose set of hyperedges is this above mentioned set. The proof relies on the three following properties of H^r_G.

Claim 19. *Any independent set of G is also an independent set of H^r_G.*

Proof. Any edge of H^r_G contains an edge of G. □

Claim 20. *Any independent set of H^r_G whose cardinality is at least r is an independent set of G.*

Proof. Any subset of $\mathrm{V}(G)$ of cardinality at least r, containing an edge e of G, also encloses hyperedges of H^r_G containing e. □

Claim 21. *The mapping that to a graph G associates the hypergraph H^r_G is computable in polynomial time.*

Proof. For a fixed r, one can enumerate the $\binom{|G|}{r} = O(|G|^r)$ r-subsets of $\mathrm{V}(G)$ and select those that contain an edge of G in polynomial time. □

Note that H^r_G and G do not always have the same independent sets.

Example 22. For $r = 3$ and $G = ([1, 4], \{\{1, 2\}, \{1, 3\}, \{2, 4\}\})$, one has:

$$H^3_G = ([1, 4], \{\{1, 2, 3\}, \{1, 2, 4\}, \{1, 3, 4\}, \{2, 3, 4\}\}).$$

The edge $\{1, 2\}$ of G is an independent set of H^3_G, but not of G.

We can now finish the proof of Theorem 18.

(i) Let us demonstrate that the restriction of r-MISH$_D$ to instances (H, k) satisfying $k \geq k_0$ is W[1]-hard for parameter k.

Without loss of generality, one can assume $k_0 \geq r$. We proceed by reduction from the restriction of 2-MISH$_D$ to instances (G, k) such that $k \geq k_0$.

Let (G, k) be an instance of 2-MISH$_D$ such that $k \geq k_0$. We build the instance (H_G^r, k) of r-MISH$_D$.

Claim 21 guarantees that this construction takes polynomial time. Moreover, Claims 19 and 20 guarantee that for any vertex set I of cardinality k, G admits I as an independent set if and only if H_G^r also admits I as an independent set. Hence, we have shown that our transformation, which preserves acceptance thresholds, is a Karp-reduction. The W[1]-hardness of r-MISH$_D$ follows from Lemma 15.

(ii) Let us prove that r-MISH is at least as hard to approximate as 2-MISH.

Assume there exists an approximation algorithm with bound $b(|H|)$ for r-MISH. We describe an approximation algorithm B with bound $b(|G|)$ for the restriction of 2-MISH to graphs such that $\alpha(G) \,/\, b(|G|) \geq r$. Then, applying Lemma 17 (with $k_0 := r$) will enable us to conclude.

Let G be an input graph satisfying $\alpha(G) \,/\, b(|G|) \geq r$. Algorithm B proceeds as follows.

(1) Compute the r-uniform hypergraph H_G^r.
(2) Compute I, an independent set of H_G^r whose cardinality is at least $\alpha(H_G^r) \,/\, b(|H_G^r|)$, and return I.

Algorithm B takes polynomial time: Step (1) because of Claim 21, and Step (2) by hypothesis. It remains to show that Algorithm B admits the correct bound, *i.e.*, that I is an independent set of G of cardinality $\#I \geq \alpha(G) \,/\, b(|G|)$. As G and H_G^r share the same vertex set, one has $|H_G^r| = |G|$. Moreover, $\alpha(G) \leq \alpha(H_G^r)$ because of Claim 19. It follows that the set I computed during Step (2) satisfies

$$\#I \geq \frac{\alpha(H_G^r)}{b(|H_G^r|)} = \frac{\alpha(H_G^r)}{b(|G|)} \geq \frac{\alpha(G)}{b(|G|)}.$$

As we restricted ourselves to input graphs G such that $\alpha(G) \,/\, b(|G|) \geq r$, Claim 20 guarantees that the set I returned by Algorithm B is an independent set of G. This concludes the proof of Theorem 18. □

Note that [32] uses the transformation $G \mapsto H_G^r$ described in the preceding proof to show a similar result about the GRAPH COLORING problem.

4.2. *Approximability of* LCS, LCUS, LCCS, *and* LCUCS

4.2.1. *Approximation algorithms*

In Propositions 23 and 25, we generalize to LCUS, LCCS, and LCUCS the approximation bounds $\sigma(X)$ and $(\min_{x \in X} |x|) \,/\, \log(\min_{x \in X} |x|)$ announced for LCS in [18] and [26], respectively. Note that an approximation bound of $\min_{x \in X} |x|$ for LCS, LCUS, LCCS, and LCUCS is trivial. Indeed, for any non-empty, finite input language X, return

- a letter common to all words in X if such a letter exists, and
- the empty word otherwise.

Proposition 23. *Each problem among* LCS, LCUS, LCCS, *and* LCUCS *admits an approximation algorithm with bound* $\sigma(X)$.

Proof. Consider the Long Run algorithm [18], which maps any non-empty, finite language X to a maximal common *unary* subsequence $\mathrm{LR}(X)$ of X. Among all letters that occur in at least a word of X, Long Run chooses a letter a that maximizes the quantity $\ell_a := \min_{x \in X} |x|_a$, and returns $\mathrm{LR}(X) := a^{\ell_a}$. This computation requires polynomial time. Moreover, $\mathrm{LR}(X)$ is a solution on instance X of the problems LCS, LCUS, LCCS, and LCUCS.

It remains to prove that Long Run has bound $\sigma(X)$, *i.e.*, that $\mathrm{LR}(X)$ has length at least $\mathrm{lcs}(X)/\sigma(X)$ (resp. $\mathrm{lcus}(X)/\sigma(X)$, resp. $\mathrm{lccs}(X)/\sigma(X)$, resp. $\mathrm{lucucs}(X)/\sigma(X)$). By Remark 5, it suffices to prove that $|\mathrm{LR}(X)| \geq \mathrm{lucucs}(X)/\sigma(X)$. For this sake, we use the following simple claim:

Claim 24. *In any word* s *(it contains* $\sigma(\{s\})$ *distinct letters), there exists a letter whose number of occurrences in* s *is at least* $|s| \,/\, \sigma(\{s\})$.

Let s be a common UC-subsequence of X of maximal length $\mathrm{lcucs}(X)$. Claim 24 implies the existence of a letter, say b, such that

$$|s|_b \geq \frac{|s|}{\sigma(\{s\})} \geq \frac{|s|}{\sigma(X)} = \frac{\mathrm{lcucs}(X)}{\sigma(X)}.$$

So, the unary word $b^{|s|_b}$ is a common subsequence of X and we obtain

$$|\mathrm{LR}(X)| \geq |b^{|s|_b}| = |s|_b \geq \frac{\mathrm{lcucs}(X)}{\sigma(X)}. \quad \square$$

Proposition 25. *Each problem among* LCS, LCUS, LCCS, *and* LCUCS *admits an approximation algorithm with bound* $\ell/\log\ell$ *where* $\ell := \min_{x\in X}|x|$.

Proof. It suffices to apply Halldòrsson's *approximating via partitioning* method [26].

(1) Select a word $x \in X$ of minimal length ℓ and factorize it under the form $x = y_1 y_2 \cdots y_p$ where
- p is a positive integer smaller than or equal to $\ell/\log\ell$,
- for any $i \in [1, p]$, y_i is a word of length $|y_i| = O(\log\ell)$.

(2) For each $i \in [1, p]$, enumerate the subsequences (resp. U-subsequences, resp. C-subsequences, resp. UC-subsequences) of y_i, and memorize those that are solutions of our problem.
(3) Among the words selected at Step (2), return one of maximal length. $\square$

4.2.2. Inapproximability of LCS, LCUS, LCCS, *and* LCUCS

In the remaining of this paper, we consider implicitly that any integer is a symbol and that all letters are drawn from the linearly ordered set $\mathbb{N}$.

Definition 26 ($S\uparrow$). Let S be a finite subset of $\mathbb{N}$. We denote by $S\uparrow$ the unique word over S that is increasing and has length $\#S$.

The following lemma enables us to obtain the results on the hardness of approximation in the unoriented case with the same argument as in the oriented case (see Theorems 29 and 35). Indeed, in a word satisfying certain conditions, a long enough increasing U-subsequence of this word also is a subsequence of it.

Lemma 27. *Let* w *be a word that can be written as the concatenation of* n *increasing words.*

(i) *Any U-subsequence of* w *that is increasing and whose length is at least* $n+1$ *is a subsequence of* w.
(ii) *Any UC-subsequence of* w *that is increasing and whose length is at least* $n+2$ *is a C-subsequence of* w.

Proof. Let $w_1, w_2, \ldots, w_n$ be n increasing words such that $w = w_1 w_2 \cdots w_n$.

(i) Let s be an increasing U-subsequence of w such that $|s| \geq n+1$. We show that s is a subsequence of w.

By hypothesis, either s is a subsequence of w or $\tilde{s}$ is a subsequence of w. So, it suffices to demonstrate that the latter eventuality is forbidden. By contradiction, assume that $\tilde{s}$ is a subsequence of $w = w_1 w_2 \cdots w_n$. Then, we can factorize $\tilde{s}$ under the form $\tilde{s} = s_1 s_2 \cdots s_n$ where s_i is a subsequence of w_i for all $i \in [1, n]$.

- Since s_i is a subsequence of the increasing word w_i, s_i is increasing.
- Besides, $\tilde{s}$ is decreasing as the mirror image of the increasing word s. Thus, s_i is decreasing as a subsequence of $\tilde{s}$.

From that we deduce that s_i is of length at most 1, since the only words that are both increasing and decreasing are reduced to a single letter or to the empty word. Hence, we have $|s| = |\tilde{s}| = |s_1| + |s_2| + \cdots + |s_n| \leq n$ which contradicts the hypothesis $|s| \geq n+1$.

(ii) Let t be an increasing UC-subsequence of w such that $|t| \geq n+2$. Let us prove that t is a C-subsequence of w.

By hypothesis, t is a U-subsequence of some conjugate w' of w. Any conjugate of w may be written as the concatenation of (at most) $n+1$ increasing words. So, Point (i) applies with $n+1$ instead of n and w' instead of w. It yields that t is a subsequence of w' and thus a C-subsequence of w. $\square$

The following example shows the bounds of Lemma 27 are tight.

Example 28. Let $w := 16\ 235\ 34\ 125$. w can be written as the concatenation of $n := 4$ increasing words: 16, 235, 34, and 125.

- 2356 is an increasing U-subsequence of w of length n, but not a subsequence of w.
- 12 356 is an increasing UC-subsequence of w of length $n+1$ but not a C-subsequence of w.

4.2.2.1. Inapproximability of LCS *and* LCUS. It is shown in [20] that LCS_D is W[2]-hard when parameterized in its acceptance threshold, k. In the following theorem, we state a somehow weaker result (LCS_D is W[1]-hard with respect to k), but we exhibit a parameter and approximation preserving reduction from 2-MISH that is simpler than the one of [20] and can be adapted to LCUS_D.

Theorem 29. *Let b be a mapping that maps $\mathbb{N}\times\mathbb{N}\times\mathbb{N}$ to the subset of reals greater than or equal to* 1.

(i) *The problems* LCS_D *and* LCUS_D *are* W[1]*-hard when parameterized in their acceptance threshold, k.*
(ii) *If either* LCS *or* LCUS *are approximable within bound b* $(\min_{x\in X}|x|, \sigma(X), \#X)$ *then* 2-MISH *is approximable within bound* $b\big(|G|, |G|, |G|^2\big)$.

Proof. First, we describe how we transform a graph G into an instance X_G of LCS (resp. of LCUS). Let G be a graph on $[1, |G|]$. To any edge $E \in \mathcal{E}(G)$, we associate a word denoted $x_{G,E}$ defined by

$$x_{G,E} := ([1,|G|]\setminus E){\uparrow}\ E{\uparrow}[2]\ E{\uparrow}[1]\ ([1,|G|]\setminus E){\uparrow}\,.$$

In $x_{G,E}$, the prefix and suffix $([1, |G|] \setminus E)\uparrow$ is the ordered set of vertices except the ones linked by E. In between, the latter are written in reverse alphabet order ($E\uparrow[2]\ E\uparrow[1]$). We can now define the instance X_G of LCS or LCUS:

$$X_G := \big\{x_{G,E}\big\}_{E\in\mathcal{E}(G)} \cup \big\{([1,|G|]{\uparrow})^p\big\}_{p\in[1,|G|^2-\#\mathcal{E}(G)]}\,.$$

In addition to the $x_{G,E}$'s ($E \in \mathcal{E}(G)$), X_G contains several powers of the word $[1, |G|]\uparrow$, which represent the whole ordered set of vertices of G. Actually, only $[1, |G|]\uparrow$ to the power one is useful to force the monotony of common U-subsequences of X_G, while the other powers permit us to set the cardinality of X_G to $|G|^2$.

Our proof relies on the following claims stating properties of X_G.

Claim 30. *Given any input graph G on $[1, |G|]$, the language X_G can be computed in polynomial time.*

Proof. Trivial. □

Claim 31. *One has* $\min_{x\in X_G}|x| = \sigma(X_G) = |G|$ *and* $\#X_G = |G|^2$.

Proof. All words of X_G are over alphabet $[1, |G|]$ and the shortest word of X_G is $[1, |G|]\uparrow$. Moreover,

$$\begin{aligned}\#X_G &= \#\big\{x_{G,E}\big\}_{E\in\mathcal{E}(G)} + \#\big\{([1,|G|]{\uparrow})^p\big\}_{p\in[1,|G|^2-\#\mathcal{E}(G)]}\\ &= \#\mathcal{E}(G) \qquad\quad + |G|^2 - \#\mathcal{E}(G)\\ &= |G|^2.\end{aligned}$$

Note that $\mathcal{E}(G)$ has cardinality at most $\binom{|G|}{2} = \frac{1}{2}|G|(|G|-1)$ and thus less than $|G|^2$. □

Claim 32. *For any independent set I of G, $I\uparrow$ is a common subsequence of X_G of length $\#I$.*

Proof. Since I is a subset of $\mathrm{V}(G) = [1, |G|]$, $I\uparrow$ is a subsequence of $([1, |G|]\uparrow)^p$ for each $p \in \mathbb{N}\setminus\{0\}$.

Let $E \in \mathcal{E}(G)$. Since I is an independent set, the two extremities of E cannot be both in I. If none occurs in I then $I\uparrow$ is a subsequence of $([1, |G|]\setminus E)\uparrow$ and thus of $x_{G,E}$. Assume a unique extremity v of $E = \{E\uparrow[1],\ E\uparrow[2]\}$ belongs to I and occurs in $I\uparrow$ at position i: $v = I\uparrow[i]$. Then, the prefix of length $i-1$ and the suffix of length $\#I - i$ of $I\uparrow$ are both subsequences of $([1, |G|]\setminus E)\uparrow$. As v can be picked up in $E\uparrow[2]\ E\uparrow[1]$, we obtain that $I\uparrow$ is a subsequence of $x_{G,E}$. This completes the proof of Claim 32. □

Claim 33. *Let s be a common subsequence of X_G. The set of letters occurring in s is an independent set of G of cardinality $|s|$.*

Proof. Since s is a subsequence of $[1, |G|]\uparrow$, s is increasing and contains $|s|$ distinct letters.

Let $E \in \mathcal{E}(G)$. As s is an increasing subsequence of $x_{G,E}$ and the extremities of $E = \{E\uparrow[1], E\uparrow[2]\}$ occur in $x_{G,E}$ in decreasing order, s cannot include both $E\uparrow[1]$ and $E\uparrow[2]$. Hence, the set of letters occurring in s is an independent set of G. □

Claim 34. *Let s be a common U-subsequence of X_G of length at least 5. The set of letters occurring in s is an independent set of G of cardinality $|s|$.*

Proof. As either s or $\tilde{s}$ is a subsequence of $[1, |G|]\uparrow$, s is either increasing or decreasing. Since the same letters occur in s and $\tilde{s}$, we can, if necessary, change s in $\tilde{s}$ to assume that s is an increasing subsequence of $[1, |G|]\uparrow$.

Let $E \in \mathcal{E}(G)$. We know that s is a U-subsequence of $x_{G,E}$ of at least 5 letters. Note that $x_{G,E}$ can be written as the concatenation of 4 increasing words: $([1, |G|] \setminus E)\uparrow$, $E\uparrow[2]$, $E\uparrow[1]$ and $([1, |G|] \setminus E)\uparrow$ again. (If $E\uparrow[1] = 1$ or if $E\uparrow[2] = |G|$, $x_{G,E}$ may be written as the concatenation of less than 4 increasing words.) By Lemma 27, s is a subsequence of $x_{G,E}$. Hence, s is a common subsequence of X_G and Claim 33 applies. □

With these properties in hand, we can now prove points (i) and (ii) of Theorem 29.

(i) Let us show that LCS_D and LCUS_D are W[1]-hard for parameter k.

Consider the mapping R that, to any instance (G, k) of 2-MISH_D, associates the ordered pair $R(G, k) := (X_G, k)$. By Claim 30, R is computable in polynomial time.

For any instance (G, k) of 2-MISH_D, Claims 32 and 33 guarantee that (G, k) is a positive instance of 2-MISH_D iff (X_G, k) is a positive instance of LCS_D. Hence, the mapping R is a Karp-reduction from 2-MISH_D to LCS_D, and since R preserves the acceptance threshold, LCS_D is W[1]-hard for parameter k.

Moreover, if k is at least 5, then (G, k) is a positive instance of 2-MISH_D iff (X_G, k) is a positive instance of LCUS_D (Claims 32 and 34). Hence, R also induces a Karp-reduction to LCUS_D from the restriction of 2-MISH_D to instances (G, k) satisfying $k \geq 5$. Therefore, Lemma 15 applies with $k_0 := 5$: LCUS_D is W[1]-hard for parameter k.

(ii) Let us now prove that LCS and LCUS are at least as hard to approximate as 2-MISH.

Let A be an approximation algorithm with bound $b(\min_{x\in X}|x|, \sigma(X), \#X)$ for LCS or LCUS. For each non-empty, finite input language X, A returns a word s that is

- either a common subsequence of X such that $b(\min_{x\in X}|x|, \sigma(X), \#X) \times |s|$ is at least $\mathrm{lcs}(X)$,
- or a common U-subsequence of X such that $b(\min_{x\in X}|x|, \sigma(X), \#X) \times |s|$ is at least $\mathrm{lcus}(X)$.

In both cases, s is a common U-subsequence of X (Remark 3) and satisfies

$$b\Big(\min_{x\in X}|x|, \sigma(X), \#X\Big) \times |s| \geq \mathrm{lcs}(X),$$

since $\mathrm{lcus}(X)$ is greater or equal to $\mathrm{lcs}(X)$ (Remark 5).

We now describe an approximation algorithm B with bound $b(|G|, |G|, |G|^2)$ for the restriction of 2-MISH to graphs G such that $\alpha(G) \,/\, b(|G|, |G|, |G|^2) \geq 5$. This will enable us to apply Lemma 17 (with $r := 2$ and $k_0 := 5$) and conclude. Let G be the input graph. Algorithm B proceeds as follows.

(1) Compute language X_G.
(2) Run Algorithm A on X_G to obtain a common U-subsequence s of X_G of length at least:

$$\frac{\mathrm{lcs}(X_G)}{b\left(\min_{x\in X_G}|x|, \sigma(X_G), \#X_G\right)}.$$

(3) Return the set I of letters occurring in s.

Algorithm B is polynomial, *i.e.*, B takes $O\big(|G|^{O(1)}\big)$ time. Indeed, Step (1) is polynomial because of Claim 30, and so is Step (2) since A is a polynomial time algorithm.

Let us now prove the approximation bound claimed for B. Claims 31 and 32 yield $b(\min_{x\in X}|x|, \sigma(X), \#X) = b\big(|G|, |G|, |G|^2\big)$ and $\mathrm{lcs}(X_G) \geq \alpha(G)$ respectively. Hence, s is of length at least:

$$\frac{\mathrm{lcs}(X_G)}{b\left(\min_{x\in X_G}|x|, \sigma(X_G), \#X_G\right)} = \frac{\mathrm{lcs}(X_G)}{b\big(|G|, |G|, |G|^2\big)} \geq \frac{\alpha(G)}{b\big(|G|, |G|, |G|^2\big)}.$$

As we restricted ourselves to input graphs G such that $\alpha(G) \,/\, b(|G|, |G|, |G|^2) \geq 5$, Claim 34 applies: the set I returned by Algorithm B is an independent set of G of cardinality $\#I = |s| \geq \alpha(G) \,/\, b\big(|G|, |G|, |G|^2\big)$. □

4.2.2.2. Inapproximability of LCCS *and* LCUCS. To demonstrate the difficulty to approximate LCCS and LCUCS, we exhibit a parameter and approximation preserving reduction from 3-MISH. This reduction resembles the one we used to obtain similar results for LCS and LCUS in Theorem 29. This section is devoted to the proof of the following theorem.

Theorem 35. *Let b be a mapping that maps $\mathbb{N} \times \mathbb{N} \times \mathbb{N}$ to the subset of reals greater than or equal to* 1.

(i) *The problems* LCCS_D *and* LCUCS_D *are* W[1]*-hard when parameterized in their acceptance threshold, k.*
(ii) *If either* LCCS *or* LCUCS *are approximable within bound $b(\min_{x\in X} |x|, \sigma(X), \#X)$ then* 3-MISH *is approximable within bound $b(|H|, |H|, |H|^3)$.*

Proof. The gadget is similar to the one of the proof of Theorem 29. Given a 3-uniform hypergraph H on $[1, |H|]$, we set

$$x_{H,E} := (([1, |H|] \setminus E)\uparrow)^2\ E\uparrow[3]\,(([1, |H|] \setminus E)\uparrow)^2\ E\uparrow[2]\,(([1, |H|] \setminus E)\uparrow)^2\ E\uparrow[1]$$

for every hyperedge $E \in \mathcal{E}(H)$ and

$$X_H := \big\{x_{H,E}\big\}_{E\in\mathcal{E}(H)} \cup \big\{([1, |H|]\uparrow)^p\big\}_{p\in[1,|H|^3-\#\mathcal{E}(H)]}.$$

The word $[1, |H|]\uparrow$ compels a conjugate of any common C-subsequence of X_H to be increasing. For any hyperedge $E = \{E\uparrow[1], E\uparrow[2], E\uparrow[3]\}$ of H, $x_{H,E}$ prevents any increasing common C-subsequence of X_H to contain simultaneously $E\uparrow[1]$, $E\uparrow[2]$, and $E\uparrow[3]$. This yields Claim 39. We include the other words in X_H, the $([1, |H|]\uparrow)^p$ with $p \in \big[2, |H|^3 - \#\mathcal{E}(H)\big]$, to set the cardinality of X_H to $|H|^3$.

As for Theorem 29, we synthesize the useful properties of X_H in five claims.

Claim 36. *Given any* 3*-uniform hypergraph H as input on $[1, |H|]$, the language X_H can be computed in polynomial time.*

Proof. Trivial. □

Claim 37. *One has* $\min_{x\in X_H} |x| = \sigma(X_H) = |H|$ *and* $\#X_H = |H|^3$.

Proof. All words of X_H are over alphabet $[1, |H|]$ and the shortest word of X_H is $[1, |H|]\uparrow$. Moreover,

$$\begin{aligned}\#X_H &= \#\big\{x_{H,E}\big\}_{E\in\mathcal{E}(H)} + \#\big\{([1, |H|]\uparrow)^p\big\}_{p\in[1,|H|^3-\#\mathcal{E}(H)]}\\ &= \#\mathcal{E}(H) \qquad\quad + |H|^3 - \#\mathcal{E}(H)\\ &= |H|^3.\end{aligned}$$

Note that $\mathcal{E}(H)$ is of cardinality at most $\binom{|H|}{3} = \frac{1}{6}|H|(|H|-1)(|H|-2)$ and thus less than $|H|^3$. □

Claim 38. *For any independent set I of H, $I\uparrow$ is a common C-subsequence of X_H of length $\#I$.*

Proof. Since I is a subset of $\text{V}(H) = [1, |H|]$, $I\uparrow$ is a subsequence of $([1, |H|]\uparrow)^p$ for each $p \in \mathbb{N} \setminus \{0\}$.

Let $E \in \mathcal{E}(H)$. Since I is an independent set of H, at least one of the three vertices linked by E does not appear in I. W.l.o.g., let us assume $E\uparrow[3] \notin I$. Then, $I\uparrow$ is a subsequence of the word

$$([1, |H|] \setminus E)\uparrow\ E\uparrow[1]\ ([1, |H|] \setminus E)\uparrow\ E\uparrow[2]\ ([1, |H|] \setminus E)\uparrow$$

and thus a C-subsequence of its conjugate

$$E\uparrow[2]\,(([1, |H|] \setminus E)\uparrow)^2\ E\uparrow[1]\ ([1, |H|] \setminus E)\uparrow,$$

which is itself a subsequence of $x_{H,E}$. Hence, $I\uparrow$ is also a C-subsequence of $x_{H,E}$, and therefore, a common C-subsequence of X_H. This completes the proof of Claim 38. □

Claim 39. *Let s be a common C-subsequence of X_H. The set of letters occurring in s is an independent set of H of cardinality $|s|$.*

Proof. There exists a conjugate of s that is a subsequence of $[1, |H|]\uparrow$. Let us change s into this conjugate. We get that s is increasing and thus $|s|$ distinct letters occur in s.

Let $E \in \mathcal{E}(H)$. Since $E\uparrow[1] < E\uparrow[2] < E\uparrow[3]$, the three vertices linked by E never appear in increasing order in any conjugate of $x_{H,E}$. Thus, at least one of them is not occurring in s. Hence, we have shown that the set of letters occurring in s is an independent set of H. □

Claim 40. *Let s be a common UC-subsequence of X_H of length at least* 11. *The set of letters occurring in s is an independent set of H of cardinality $|s|$.*

Proof. There exists either a conjugate of s or a conjugate of $\tilde{s}$ that is a subsequence of $[1, |H|]\uparrow$. Hence, we can assume that s is a subsequence of $[1, |H|]\uparrow$ and thus is increasing.

Let $E \in \mathcal{E}(H)$. We know that s is a UC-subsequence of $x_{H,E}$ of at least 11 letters. Note that $x_{H,E}$ can be written as the concatenation of 9 increasing words:

- twice $([1, |H|] \setminus E)\uparrow$,
- $E\uparrow[3]$,
- twice $([1, |H|] \setminus E)\uparrow$,
- $E\uparrow[2]$, and again
- twice $([1, |H|] \setminus E)\uparrow$ followed by
- $E\uparrow[1]$.

By Lemma 27(ii), s is a C-subsequence of $x_{H,E}$.

Hence, s is a common C-subsequence of X_H and one can apply Claim 39 to conclude the proof of Claim 40. □

We now prove points (i) and (ii) of Theorem 35 from the preceding claims, in the same way as we deduced points (i) and (ii) of Theorem 29 from Claims 30–34.

(i) Let us show that LCCS_D and LCUCS_D are W[1]-hard for parameter k.

Consider the mapping R that, to any instance (H, k) of 3-MISH_D, associates the ordered pair $R(H, k) := (X_H, k)$.

- By Claims 36, 38 and 39, R is a Karp-reduction from 3-MISH_D to LCCS_D.
- By Claims 36, 38 and 40, R induces a Karp-reduction to LCUCS_D from the restriction of 3-MISH_D to instances (H, k) satisfying $k \geq 11$.

Hence, Theorem 18(i) (applied with $r := 3$) ensures that LCCS_D and LCUCS_D are W[1]-hard for parameter k.

(ii) Let us now prove that LCCS and LCUCS are at least as hard to approximate as 3-MISH.

Let A be an approximation algorithm with bound $b(\min_{x\in X}|x|, \sigma(X), \#X)$ for LCCS or LCUCS.

Consider the following algorithm B. Let H be a 3-uniform hypergraph.

(1) Compute language X_H.

(2) Run Algorithm A on X_H to obtain a common UC-subsequence s of X_H of length at least:

$$\frac{\text{lccs}(X_H)}{b\left(\min_{x\in X_H}|x|, \sigma(X_H), \#X_H\right)}.$$

(3) Return the set I of letters occurring in s.

Relying on Claims 36–38 and 40, it is easy to prove that B is an approximation algorithm with bound $b(|G|, |G|, |G|^3)$ for the restriction of 3-MISH to hypergraphs H such that $\alpha(H) \,/\, b(|H|, |H|, |H|^2) \geq 11$. Then, applying Lemma 17 (with $r := 3$ and $k_0 := 11$) yields the desired statement and concludes the proof. □

5. Conclusion

Our investigation provides the first hardness and approximability results concerning LCS for cyclic and unoriented strings.

Table 2
Parameterized complexity of LCUS, LCCS and LCUCS

$\#X$	$\sigma(X)$	k	LCUS_D / LCCS_D / LCUCS_D
–	≤ 2	–	NP-complete (Theorem 14)
Bounded	–	–	Polynomial (Proposition 7)
Parameter	≤ 2	–	W[1]-hard (Theorem 14)
–	Parameter	Parameter	F.P.T. (see Section 2.3.5.1)
–	–	Parameter	W[1]-hard (Theorems 29 and 35)
Parameter	–	Parameter	Open
Parameter	Parameter	–	W[t]-hard $\forall t \geq 1$ (Theorem 14)

5.1. Summary of our results

5.1.1. Parameterized complexity

Theorem 18(i) shows that, for any $r \in \mathbb{N} \setminus \{0, 1, 2\}$, r-MISH_D is W[1]-hard with respect to its acceptance threshold. Note that 2-MISH_D is W[1]-complete for this parameter [25].

Our results concerning the complexity of LCUS_D, LCCS_D and LCUCS_D are summarized in Table 2.

5.1.2. Approximability

Håstad has shown that 2-MISH is not approximable within bound $|G|^{1-\epsilon}$, unless NP = ZPP [28]. Hence, Jiang and Li's reduction from 2-MISH to LCS ensures that LCS is hard to approximate within

$$\max\left\{\min_{x\in X}|x|^{1-\epsilon}, \sigma(X)^{1-\epsilon}, (\#X)^{1-\epsilon}\right\}$$

[18] (see also Section 2.3.5.2).

Moreover, we deduce from Theorem 29(ii), that LCUS is hard to approximate within bound

$$\max\left\{\min_{x\in X}|x|^{1-\epsilon}, \sigma(X)^{1-\epsilon}, (\#X)^{0.5-\epsilon}\right\}.$$

On the other hand, Theorem 18(ii) enables us to generalize Håstad's result: for any $r \in \mathbb{N} \setminus \{0, 1\}$, r-MISH is hard to approximate within $|H|^{1-\epsilon}$. Hence, by Theorem 35(ii), LCCS and LCUCS are hard to approximate within bound

$$\max\left\{\min_{x\in X}|x|^{1-\epsilon}, \sigma(X)^{1-\epsilon}, (\#X)^{1/3-\epsilon}\right\}.$$

5.2. Open questions

It is shown in [20] that LCS_D is W[1]-complete for parameter $(\#X, k)$, but the complexity of LCUS, LCCS and LCUCS for this parameter is unknown.

The problem 2-MISH gave rise to numerous publications, some of which are referenced here, but fewer works concern MISH and r-MISH with $r \in \mathbb{N} \setminus \{0, 1, 2\}$. Specifically, both the existence of a $t \in \mathbb{N} \setminus \{0\}$ such that 3-MISH_D parameterized in its acceptance threshold k belongs to W[t], and the existence of an approximation algorithm for 3-MISH with bound $o(|H|/\log|H|)$ are open.

- The existence of a positive real constant δ, such that one problem among LCS, LCUS, LCCS or LCUCS is approximable within bound $(\#X)^{\delta}$ is open.
- For any $\sigma \in \mathbb{N}$, we demonstrated (Proposition 23) that the problems LCS, LCUS, LCCS and LCUCS restricted to instances X for which $\sigma(X) \leq \sigma$ admit an approximation algorithm with bound σ, but the existence of a Polynomial Time Approximation Scheme requires further studies.

Acknowledgement

Both authors are supported by grants from the ACI IMPBio project "REPEVOL".

References

[1] D. Sankoff, J.B. Kruskal (Eds.), Time Warps, String Edits and Macromolecules: The Theory and Practice of Sequence Comparison, 2nd edition, CSLI Publications, 1999.
[2] D. Gusfield, Algorithms on Strings, Trees, and Sequences, Computer Science and Computational Biology, Cambridge University Press, 1997.
[3] B. Alberts, D. Bray, J. Lewis, M. Raff, K. Roberts, J.D. Watson, Molecular Biology of the Cell, Garland, New York, 1983.
[4] M.D. Adams, J.M. Kelley, J.D. Gocayne, M. Dubnick, M.H. Polymeropoulos, H. Xiao, C.R. Merril, A. Wu, B. Olde, R.F. Moreno, et al., Complementary DNA sequencing: Expressed sequence tags and human genome project, Science 252 (5013) (1991) 1651–1656.
[5] H. Bunke, U. Buehler, Applications of approximate string matching to 2D shape recognition, Pattern Recognition 26 (12) (1993) 1797–1812.
[6] G. Peris, A. Marzal, Fast cyclic edit distance computation with weighted edit costs in classification, in: Proceedings of the 16th International Conference on Pattern Recognition, ICPR'02, vol. IV, IEEE Computer Society, 2002, pp. 184–187.
[7] A. Marzal, G. Peris, Normalized cyclic edit distances: An efficient algorithm, in: Proceedings of the 10th Conferencia de la Asociación Española Para la Inteligencia Artificial, CAEPIA'03, in: Lecture Notes in Computer Science, vol. 3040, Springer-Verlag, 2004, pp. 435–444.
[8] D.S. Hirschberg, Algorithms for the longest common subsequence problem, Journal of the Association for Computing Machinery 24 (4) (1977) 664–675.
[9] W.J. Masek, M.S. Paterson, A faster algorithm computing string edit distances, Journal of Computer and System Sciences 20 (1) (1980) 18–31.
[10] E.W. Myers, An $O(nd)$ difference algorithm and its variations, Algorithmica 1 (2) (1986) 251–266.
[11] M. Crochemore, G.M. Landau, M. Ziv-Ukelson, A subquadratic sequence alignment algorithm for unrestricted scoring matrices, SIAM Journal on Computing 32 (6) (2003) 1654–1673.
[12] M. Maes, On a cyclic string-to-string correction problem, Information Processing Letters 35 (2) (1990) 73–78.
[13] J.P. Schmidt, All highest scoring paths in weighted grid graphs and its application to finding all approximate repeats in strings, SIAM Journal on Computing 27 (4) (1998) 972–992.
[14] G.M. Landau, E.W. Myers, J.P. Schmidt, Incremental string comparison, SIAM Journal on Computing 27 (2) (1998) 557–582.
[15] S.Y. Itoga, The string merging problem, BIT Numerical Mathematics 21 (1) (1981) 20–30.
[16] D. Maier, The complexity of some problems on subsequences and supersequences, Journal of the Association for Computing Machinery 25 (2) (1978) 322–336.
[17] K. Pietrzak, On the parameterized complexity of the fixed alphabet shortest common supersequence and longest common subsequence problems, Journal of Computer and System Sciences 67 (4) (2003) 757–771.
[18] T. Jiang, M. Li, On the approximation of shortest common supersequences and longest common subsequences, SIAM Journal on Computing 24 (5) (1995) 1122–1139.
[19] H.L. Bodlaender, R.G. Downey, M.R. Fellows, M.T. Hallett, H.T. Wareham, Parameterized complexity analysis in computational biology, Computer Applications in the Biosciences (CABIOS) 11 (1) (1995) 49–57.
[20] H.L. Bodlaender, R.G. Downey, M.R. Fellows, H.T. Wareham, The parameterized complexity of sequence alignment and consensus, Theoretical Computer Science 147 (1–2) (1995) 31–54.
[21] K.-J. Räihä, E. Ukkonen, The shortest common supersequence problem over binary alphabet is NP-complete, Theoretical Computer Science 16 (2) (1981) 187–198.
[22] V.G. Timkovskii, Complexity of common subsequence and supersequence problems and related problems, Cybernetics 25 (1990) 565–580.
[23] M. Middendorf, More on the complexity of common superstring and supersequence problems, Theoretical Computer Science 125 (2) (1994) 205–228.
[24] T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein, Introduction to Algorithms, 2nd edition, MIT Press and McGraw-Hill, 2001.
[25] R.G. Downey, M.R. Fellows, Parameterized Complexity, Monographs in Computer Science, Springer, 1999.
[26] M.M. Halldòrsson, Approximations of weighted independent set and hereditary subset problems, Journal of Graph Algorithms and Applications 4 (1) (2000) 1–16.
[27] R. Boppana, M.M. Halldòrsson, Approximating maximum independent sets by excluding subgraphs, BIT Numerical Mathematics 32 (2) (1992) 180–196.
[28] J. Håstad, Clique is hard to approximate within $n^{1-\epsilon}$, Acta Mathematica 182 (1999) 105–142.
[29] L. Engebretsen, J. Holmerin, Towards optimal lower bounds for clique and chromatic number, Theoretical Computer Science 299 (1–3) (2003) 537–584.
[30] M. Lothaire (Ed.), Combinatorics on Words, 2nd edition, Cambridge Mathematical Library, Cambridge University Press, 1997.
[31] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, M. Protasi, Complexity and Approximation, 2nd edition, Springer-Verlag, 2003.
[32] M. Krivelevich, B. Sudakov, Approximate coloring of uniform hypergraphs, Journal of Algorithms 49 (1) (2003) 2–12.